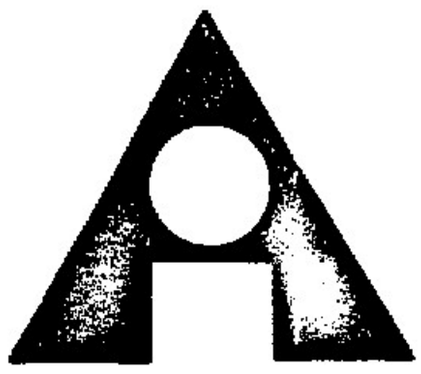
LEARNING AND GENERALIZATION OF CHARACTERISTIC DESCRIPTIONS: EVALUATION CRITERIA AND COMPARATIVE REVIEW OF SELECTED METHODS

by

T. G. Dietterich
R. S. Michalski

Proceedings of the Sixth International Joint Conference on Artificial Intelligence, Tokyo, pp. 223-231, August 20-23,1979.

1979-6



IJCAI-79

第６回人工知能国際会議

Proceedings of the Sixth International
Joint Conference on Artificial Intelligence
Tokyo, August 20–23, 1979
Volume One

LEARNING AND GENERALIZATION OF CHARACTERISTIC DESCRIPTIONS: EVALUATION CRITERIA AND COMPARATIVE REVIEW OF SELECTED METHODS

Thomas G. Dietterich
Department of Computer Science
University of Illinois
Urbana, Illinois 61801

Ryszard S. Michalski*
Department of Computer Science
University of Illinois
Urbana, Illinois 61801

Some recent work in the area of learning structural descriptions from examples is reviewed in light of the need in many diverse disciplines for programs which can perform conceptual data analysis. Such programs describe complex data in terms of logical, functional, and causal relationships which cannot be discovered using traditional data analysis techniques. Various important aspects of the problem of learning structural descriptions are examined and criteria for evaluating current work is presented. Methods published by Buchanan, et.al.[1-3,20], Hayes-Roth [6-9], and Vere [22-25], are analyzed according to these criteria and compared to a method developed by the authors. Finally some goals are suggested for future research.

1. INTRODUCTION

1.1 Motivation and Basic Concepts

There are many problem areas where large volumes of data are generated about a class of objects, the behavior of a system, a process, etc. Scientists in fields as diverse as agriculture, chemistry, and psychology are faced with the need to analyze such data in order to detect regularities and common patterns. Traditional tools for data analysis include various statistical techniques, curve-fitting techniques, numerical taxonomy, etc. These methods, however, are often not satisfactory because they impose an overly restrictive mathematical framework on the scope of possible solutions. For example, statistical methods describe the data in terms of probability distribution functions placed on random variables. As a result, the types of patterns which they can discover are limited to those which can be expressed by placing constraints upon the parameters of various probability distribution functions. Because of the mathematical frameworks upon which they are based, traditional methods cannot detect conceptual patterns such as the logical, causal, or functional relationships that are typical of descriptions produced by humans. This is a well-known problem in AI, namely that a system in order to learn something must first be able to express it. The solution requires introducing more powerful representations for hypotheses and developing corresponding techniques of data analysis and pattern discovery. Work done in AI and related areas on computer induction and learning structural descriptions from examples has laid the groundwork for researh in this area. This is not accidental, because, as Michie [18] has pointed out, the development of systems which deal with problems in human conceptual terms is a fundamental characteristic of AI research.

In this paper, we examine some of the recent work in AI on the subject of learning and generalization of structural descriptions. In particular, we will review four recent methods of inductive generalization: Buchanan et. al., Hayes-Roth, Vere, and our own work (Earlier well-known work by Winston was recently reviewed by Knapman [11]). We also outline some goals for research in this area. Attention is given primarily to the simplest form of generalization, namely the maximally specific conjunctive statements which characterize a single set of input events (called for short, conjunctive generalizations). The reason for this choice is that most work done in this area is addressing this, quite restricted, subject. Many of the researchers whose work we review in this paper have done work on other aspects of machine learning including generalization using negative examples (Vere, Michalski) and developing discriminant descriptions of several classes of objects (Michalski). Due to space limitations, we have been unable to include these topics in this paper. Instead, thhese contributions are mentioned in the sections concerning extensions. We begin the analysis by first discussing several important aspects of the problem of learning conceptual descriptions:

- types of descriptions: characteristic versus discriminant
- forms of descriptions
- types of generalization processes involved in generalizing descriptions (rules of generalization)
- constructive versus non-constructive induction
- general versus problem-oriented methods of induction.

1.2 Types of Descriptions

We distinguish between characteristic and discriminant descriptions [16]. A characteristic description is a description of a single set of objects (examples, events) which is intended to discriminate that set of objects from all other possible objects. For example, a characteristic description of the set of all tables would discriminate any table from all things which are non-tables. Psychologists consider this problem under the name of concept formation (e.g. Hunt [10]). Since it is impos-

* The authors gratefully acknowledge the support of NSF under grant MCS-76-22940.

sible to examine all other possible objects, a characteristic description is usually developed by specifying all characteristics which are true for all known objects of the class (positive examples). Alternatively, in some problems there are available so-called "near misses" which can be used to more precisely circumscribe the given class.

A discriminant description is a description of a single class of objects in the context of a fixed set of other classes of objects. It states only those properties of objects in the class under consideration which are necessary to distinguish them from the objects in the other classes. A characteristic description can be viewed as a discriminant description in which the given class is discriminated against infinitely many alternative classes.

In this paper we restrict ourselves to the problem of determining characteristic descriptions. The problem of determining discriminant descriptions has been studied by Michalski and his collaborators [13-17]).

1.3 Forms of Descriptions

Descriptions, either characteristic or discriminant, may take several forms. In this paper we concentrate on generalizations in conjunctive form. Other forms include disjunctions, exceptions, production rules of various types, hierarchical and multilevel descriptions, semantic nets, and frames.

1.4 Generalization Rules

The process of inducing a general description from examples can be viewed as a process of applying certain generalization rules to the initial descriptions to transform them into more general output descriptions. This viewpoint permits one to characterize various methods of induction by specifying the rules of generalization which they use. Below is a brief review of various generalization rules based on the paper [17].

i) Dropping Condition Rule. If a description is viewed as a conjunction of conditions which must be satisfied, then one way to generalize it is to drop one or more of these conditions. For example:

$$red(x) \wedge big(x) \;\; |< \;\; red(x)$$

(this reads: "the description 'xs which are red and big' can be generalized to the description 'xs which are red"; |< denotes the generalization operator)

ii) Turning Constants to Variables Rule. If we have two or more descriptions, each of which refers to a specific object (in a set to be characterized), we can generalize these by creating one description which contains a variable in place of the specific object:

$$\begin{matrix} tall(Fred) \wedge man(Fred) \\ tall(Jim) \wedge man(Jim) \end{matrix} \;\; |< \;\; \forall x \; tall(x) \wedge man(x)$$

assuming that the value set of x is {Fred, Jim, ... }. 'x' can be interpreted as representing 'a person from the group under consideration.'

These first two rules of generalization are the rules most commonly used in the literature on computer induction. Both rules can, however, be viewed as special cases of the following rule.

iii) Generalizing by Internal Disjunction Rule. A description can be generalized by extending the set of values that a descriptor (i.e. variable, function, or predicate) is permitted to take on in order that the description is satisfied. This process involves an operation called the internal disjunction. For example:

$$\begin{matrix} shape(x,square) \\ shape(x,triangle) \end{matrix} \;\; |< \;\; shape(x,(square \text{ or } triangle \text{ or } rectangle))$$

where statements on the left of |< describe some single objects in a class, and the statement on the right is a plausible generalization.

Using the notation of variable-valued logic system VL_{21} [17] this rule can be expressed somewhat more compactly:

$$\begin{matrix} [shape(x)=square] \\ [shape(x)=triangle] \end{matrix} \;\; |< \;\; [shape(x)=square, triangle, rectangle]$$

The ',' in the expression on the right of the |< denotes the internal disjunction. Although it may seem at first glance that the internal disjunction is just a notational abbreviation, this operation appears to be one of the fundamental operations people use in generalizing descriptions.

In general this rule can be expressed:

$$W[L = R1] \;\; |< \;\; W[L = R2]$$

where W is some condition and where R1 and R2 are sets of values linked by internal disjunction, and R1 ⊄ R2.

There are two important special cases of this rule. First, when the descriptor involved takes on values which are linearly ordered (a linear descriptor) and the second when the descriptor takes on values which represent concepts at various levels of generality (a structured descriptor).

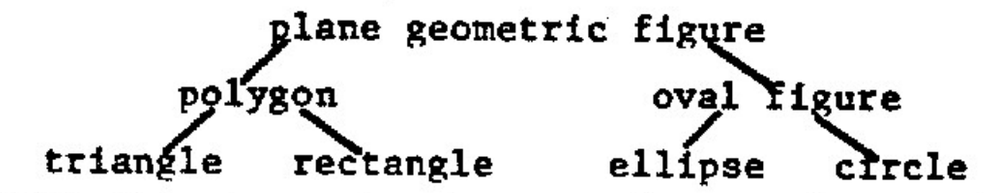
In the case of a linear descriptor we have:

iv) Closing Interval Rule. For example, suppose two objects of the same class have all the same characteristics except that they have different sizes, a and b. Then, it is plausible to hypothesize that all objects which share these characteristics but which have sizes between a and b are also in this class.

$$\begin{matrix} W\,[size(x1)=a] \\ W\,[size(x2)=b] \end{matrix} \;\; |< \;\; W\,[size(x) = a..b]$$

In the case of structured descriptors we have:

v) Climbing Generalization Tree Rule. Suppose the value set of the shape descriptor is the tree of concepts:



With this tree structure, values such as triangle and rectangle can be generalized by climbing the generalization tree:

$$\begin{matrix} [shape(x)=rectangle] \\ [shape(x)=triangle] \end{matrix} \;\; |< \;\; [shape(x)=polygon]$$

1.5 Constructive Induction

Most methods of induction produce descriptions which involve the same descriptors which were present in the initial data. These methods operate by selecting descriptors from the input data and putting them into a form which is an appropriate generalization. Such methods perform non-constructive induction. A method performs constructive induction if it includes mechanisms which can generate new descriptors not present in the input data. These new descriptors are generated by applying rules of

constructive induction. Such rules may be written as procedures or as production rules and may be based on general knowledge or on problem-oriented knowledge (for examples of constructive generalization rules see [17]). Constructive induction rules can interpret the input data in terms of knowledge about the problem domain. Frequently, the solution to a problem is dependent upon finding the proper description for the problem; as in the mutilated checkerboard problem. An inductive program should contain facilities for constructive induction including a library of general constructive induction rules. The user should be able to suggest new rules for the program to examine. In order to activate those rules which would be most useful, the program must be able to efficiently search the space of possible constructive induction rules.

Programs which perform constructive induction are more likely to find useful and interesting patterns in complex data since they have the ability to examine the data using many different representations.

1.6 General versus Problem-oriented Methods

It is a common view that general methods of induction, although mathematically elegant and theoretically applicable to many problems, are in practice very inefficient and rarely lead to any interesting solutions. This opinion seems to have lead certain workers to abandon (at least temporarily) work on general methods and concentrate on some specific problem (e.g., Buchanan, et. al. [1,2,3] or Lenat [12]). This approach often leads to interesting and practical solutions. On the other hand, it is often difficult to extract general principles of induction from such problem-specific work. It is also difficult to apply such special-purpose programs to new areas.

An attractive possibility for solving this dilemma is to develop methods which incorporate various general principles of induction (including constructive induction) together with mechanisms for using exchangeable packages of problem-specific knowledge. In this way a general method of induction, provided with an appropriate package of knowledge, could be both easily applicable to different problems and also efficient and practically useful. This idea underlies the development of the INDUCE programs [14,17,4].

2. COMPARATIVE REVIEW OF SELECTED METHODS

2.1 Evaluation Criteria

We evaluate the selected methods of induction in terms of several criteria considered especially important in view of the remarks in section 1.

i) Adequacy of the representation language. The language used to represent input data and output generalizations determines to a large extent the quality and usefulness of the output descriptions. Although it is difficult to assess the adequacy of a representation language out of the context of some specific problem, recent work in AI has shown that languages which treat all phenomena uniformly must sacrifice descriptive precision. For example, researchers who are attempting to build natural-language systems prefer the richer knowledge representations such as frames and semantic nets (with their tremendous variety of syntactic forms) to more uniform and less structured representations such as attribute-value lists and PLANNER-style databases. In our own work on inductive learning, we have chosen to use the representation language VL_{21} (see below) which has a wider variety of syntactic forms than our earlier language VL_1. Although languages with many syntactic forms do provide greater descriptive precision, they also make the induction process more complex. In order to control this complexity, a compromise must be sought between uniformity and richness of forms. In the evaluation of each method, a review of the operators and syntactic forms of each description language is provided.

ii) Rules of generalization implemented. The generalization rules implemented in each algorithm are listed.

iii) Computational efficiency. The exact analysis of the computational efficiency of these algorithms is very difficult due both to the inherent complexity of the algorithms and to the lack of precise formulations of the algorithms in available publications. However, it seems useful to have some data comparing the efficiency of these algorithms even if that data is approximate and based on hand-simulations. To get some indication of the efficiency we measure the total number of description generations or comparisons required by each method to perform a test example (see Fig. 1). We also measure the ratio of the number of output conjunctive generalizations to the total number of generalizations examined on this example. Since these numbers are derived from only one example, it is not appropriate to draw strong conclusions from them concerning the general performance of the algorithms. Our conclusions are based primarily on the general behavior of the algorithms.

iv) Flexibility and extensibility. Mere conjunctive characteristic generalizations are not particularly useful for conceptual data analysis because of their limited format and their lack of formal mechanisms for handling errors in the input data. It is important in evaluating these algorithms to consider the ease with which each method could be extended to

a) discover descriptions with forms other than conjunctive generalizations (see section 1.3),

b) include mechanisms which facilitate the detection of errors in the input data,

c) provide a general facility for incorporating domain-specific knowledge into the induction process as an exchangeable package (Ideally, the domain-specific knowledge should be isolated from the general-purpose inductive process.), and

d) perform constructive induction.

It is difficult to assess the flexibility and extensibility of the algorithms presented here. We base our evaluation on the general approaches of the methods and on extensions which have already been made to them.

In the following sections, we describe each method by presenting the description language used, sketching the underlying algorithm, and evaluating the method in terms of the above criteria. Each method will be illustrated using the test example shown in Fig. 1.

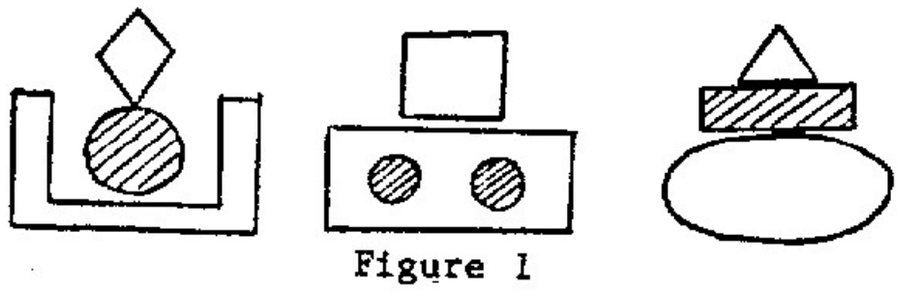
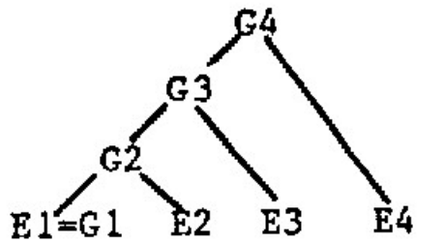


Figure 1

2.2 Data-driven Methods: Hayes-Roth and Vere.

Methods can be divided into bottom-up (data-driven), top-down (model-driven), and mixed methods. Bottom-up methods generalize the input events pairwise until the final conjunctive generalization is computed:



G2 is the set of conjunctive generalizations of E1 and E2. Gi is the set of conjunctive generalizations obtained by taking each element of Gi-1 and generalizing it with Ei.

We consider here only the methods described by Hayes-Roth and Vere. Other bottom-up methods include the candidate elimination approach described by Mitchell [19] and the Uniclass method described by Stepp [21].

2.2.1 Hayes-Roth: Program SPROUTER [6-9]

Hayes-Roth uses the term maximal abstraction or interference match for maximally specific conjunctive generalization. He uses parameterized structural representations (PSRs) to represent both the input events and their generalizations. For example, consider the two events described in Fig. 2:

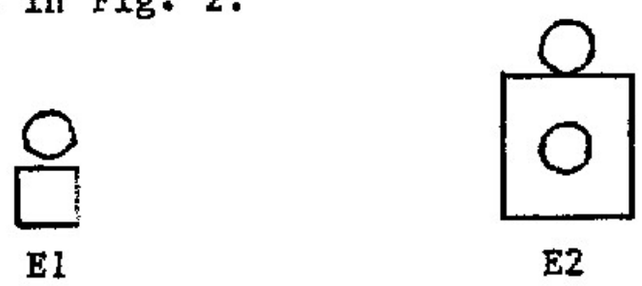


Figure 2

The PSRs for these could be:

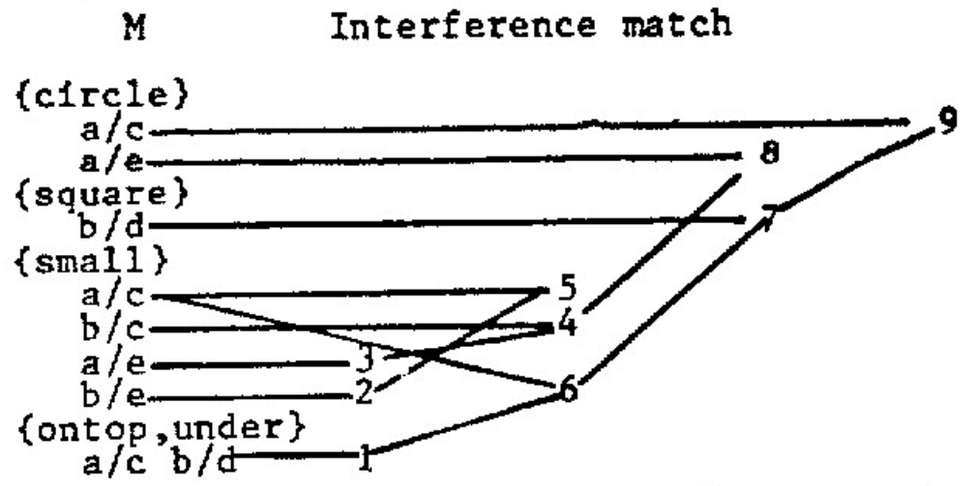
```
E1: {{circle:a}{square:b}{small:a}
     {small:b}{ontop:a, under:b}}
E2: {{circle:c}{square:d}{circle:e}
     {small:c}{large:d}{small:e}
     {ontop:c, under:d}
     {inside:e, outside:d}}
```

The expressions such as {small:a} are case frames made up of case labels (small, circle, etc.) and parameters (a, b, c, d). The PSR can be interpreted as a conjunction of predicates of the form small(a) where the parameters are existentially quantified variables which are assumed to be distinct.

The interference match attempts to find the longest one-to-one match of parameters and case frames (i.e., the longest common subexpression). This is accomplished in two steps. First the case relations in E1 and E2 are matched in all possible ways to obtain the set M. Two case relations match if all of their case labels match. Each element of M is a case relation and a list of parameter correspondences which permit that case relation to match in both events:

```
M = {{circle:((a/c)(a/e))}{square:((b/d))}
     {small:((a/c)(b/c)(a/e)(b/e))}
     {ontop,under:((a/c b/d))}}
```

The second step involves selecting a subset of the parameter correspondences in M such that all parameters can be bound consistently. This is conducted by a breadth-first search of the space of possible bindings with pruning of unpromising nodes. The search can be visualized as a node-building process. Here is one such (pruned) search:



The nodes are numbered in order of generation. One at a time, a node is examined and joined with all other consistent nodes which have already been examined. The nodes 5, 8, and 9 are conjunctive generalizations. Node 9 binds a to c (to give 1) and b to d (to give 2) to produce the conjunction:

```
{{circle:1}{square:2}{small:1}
 {ontop:1, under:2}}
```

The node-building process is guided by computing a utility value for each candidate node to be built. The nodes are pruned by setting an upper limit on the total number of possible nodes and pruning nodes of low utility when that limit is reached.

Evaluation:

i) Representational adequacy. The algorithm discovers the following conjunctive generalizations of the example in Fig. 1:

1. {{ontop:1, under:2}{medium:1}{clear:1}}
   There is a medium clear object ontop of something.
2. {{ontop:1, under:2}{medium:1}{large:2}{clear:2}}
   There is a medium object ontop of a large, clear object.
3. {{medium:1}{clear:1}{large:3}{clear:3}{shaded:2}}
   There is a medium sized clear object, a large sized clear object, and a shaded object.

PSRs provide two symbolic forms: parameters and case labels. The case labels can express ordinary predicates and relations easily. Symmetric relations may be expresed by using the same label twice as in {same!size:a, same!size:b}. The only operator is the conjunction. The language has no disjunction or internal disjunction. As a result, the fact that the top element in Fig. 1 is always either a square or a diamond cannot be discovered.

ii) Rules of generalization. The method uses the dropping condition and turning constants to variables rules.

iii) Computational efficiency. On our test example, the algorithm requires 22 comparisons and generates 20 candidate conjunctive generalizations of which 6 are retained. This gives a figure of 6/20 or 30% for computational efficiency. Four separate interference matches are required since the first match of E1 and E2 produces three possible conjunctive generalizations.

iv) Flexibility and extensibility. Hayes-Roth has indicated (personal communication) that this method has been extended to produce disjunctive generalizations and to detect errors in data. Hayes-Roth has applied this method to various problems in the design of the speech understanding system Hearsay II. Howev-

er, no facility has been developed for incorporating domain-specific knowledge into the generalization process.

Also, no facility for constructive induction has been incorporated although Hayes-Roth has developed a technique for converting a PSR to a lower-level finer-grained uniform PSR. This transformation permits the program to develop descriptions which involve a many-to-one binding of parameters.

2.2.2 Vere: Program Thoth [22-25]

Vere uses the term maximal conjunctive generalization or maximal unifying generalization to denote the maximally specific conjunctive generalization. Each event is represented as a conjunction of literals. A literal is a parenthesized list of constants called terms. For example, the objects in Fig. 2 would be described:

E1: (circle a)(square b)(small a)(small b)
(ontop a b)
E2: (circle c)(square d)(circle e)
(small c)(large d)(small e)
(ontop c d)(inside e d)

Although these resemble Hayes-Roth's PSRs, they are quite different. There are no distinguished symbols. All terms are treated uniformly.

The algorithm operates in four steps. First, the literals in each of the two events to be generalized are matched in all possible ways to generate the set of matching pairs MP. Two literals match if they contain the same number of constants and they share a common term in the same position. For the example of Fig. 2,

MP= { ((circle a)-(circle c)),
((circle a)-(circle e)),
((square b)-(square d)),
((small a)-(small c)),
((small a)-(small e)),
((small b)-(small c)),
((small b)-(small e)),
((ontop a b)-(ontop c d)) }

The second step involves selecting all possible subsets of MP such that no single literal of one event is paired with more than one literal in another event. Each of these subsets eventually forms a new generalization of the original events.

In the third step, each subset of matching pairs selected in step 2 is extended by adding to the subset additional pairs of literals which did not previously match. A new pair p is added to a subset S of MP if each literal in p is related to some other pair q in S by a common constant in a common position. For example, if S contained the pair ((square b)-(square d)) then we could add to S the pair ((ontop a b)-(inside e d)) because the third element of (ontop a b) is the second element of (square b) and the third element of (inside e d) is the second element of (square d) (Vere calls this a 3-2 relationship). We continue adding new pairs until no more can be added.

In step 4 the resulting set of pairs is converted into a new conjunction of literals by merging each pair to form a single literal. Constants which do not match are turned into new constants which may be viewed as variables. For example, ((circle a)-(circle c)) would be converted to (circle 1).

Evaluation:

i) Representational adequacy. When applied to the test example (Fig. 1) this algorithm produces many generalizations. A few of the significant ones are listed here:

1. (ontop 1 2)(medium 1)(large 2)(clear 2)(clear 3)(shaded 4)(5 4)
There is a medium object on top of a large clear object. Another object is clear. There is a shaded object. (Note also the vacuous relationship 5 derived from unifying circle and triangle).
2. (ontop 1 2)(clear 1)(medium 1)(9 1) (5 3 4)(shaded 3)(7 3)(6 3)(clear 4) (large 4)(8 4)
There is a medium, clear object on top of some other object and there are two objects related in some way (5) such that one is shaded and the other is large and clear. (Note the vacuous relationships 6, 7, 8, and 9).
3. (ontop 1 2)(medium 1)(clear 2)(large 2)(5 2) (shaded 3)(7 3)(clear 4)(6 4)
There is a medium object on top of a large clear object. There is a shaded object and there is a clear object. (Note the vacuous relationships 5, 6, and 7).

The representation is very general. By convention the first symbol of a literal can be interpreted as a predicate symbol. The algorithm, however, treats all constants uniformly. This creates difficulties. For instance the algorithm generates vacuous literals in certain situations. Literals can be formed by pairing (red x) with (big y) to produce meaningless generalizations. One advantage of this relaxation of semantic constraints is that the program can discover conjunctive generalizations involving a many-to-one binding of variables.

The language contains only a conjunction operator. No disjunction or internal disjunction is included.

ii) Rules of generalization. The algorithm implements the dropping condition rule and the turning constants to variables rule.

iii) Computational efficiency. From the published articles [22-25] it is not clear how to perform step 2. The space of possibilities is very large and an exhaustive search could not possibly give the computation times which Vere has published. It would be interesting to find out what heuristics are being used to guide the search.

iv) Flexibility and extensibility. Vere has published algorithms which discover descriptions with disjunctions [24] and exceptions [25]. He has also developed techniques to generalize relational production rules [23,24]. The method has been demonstrated using the traditional AI toy problems of IQ analogy tests and blocks-world sequences. A facility for using background information to assist the induction process has also been developed. It uses a spreading activation technique to extract relevant relations from a knowledge base and add them to the input examples prior to generalizing them. Since the method has been extended to discover disjunctions and exceptions, it would be expected that the method could also operate in noisy environments.

2.3 Model-driven Methods: Buchanan et. al., and Michalski.

Model-driven methods search a set of possible generalizations in an attempt to find a few "best" hypotheses which satisfy certain requirements. The two methods discussed here search for a small number of conjunctions which together cover all of the input events. The search proceeds by choosing as the initial working hypothesis some starting point in the partially ordered set of all possible descrip-

tions. If the working hypotheses satisfy certain termination criteria, then the search halts. Otherwise, the current hypotheses are modified by slightly generalizing or specializing them. These new hypotheses are then checked to see if they satisfy the termination criteria. The process of modifying and checking continues until the criteria are met. Top-down techniques typically have better noise immunity and can easily be extended to discover disjunctions. The principal disadvantage of these techniques is that the working hypotheses must repeatedly be checked to determine whether they subsume all of the input events.

2.3.1 Buchanan, et. al.: Program Meta-DENDRAL [1-3,20]

The algorithm which we describe here is taken from the RULEGEN program (part of the Meta-DENDRAL system). Meta-DENDRAL was designed to discover cleavage rules to explain mass spectrometry data. The descriptive language is based on the ball-and-stick model of chemical molecules. Each input event is a bond environment which describes some portion of a molecule. The environment is represented by a graph of the atoms in the molecule with four descriptors attached to each atom and forms the left hand side of a cleavage rule. The right hand side of the rule predicts a cleavage based on the existence in a molecule of the left-hand side of the rule (breakbond (**) indicates that the ** bond is predicted to be broken). A typical cleavage rule (with atoms w, x, y, and z) is:

LEFT-HAND SIDE (BOND ENVIRONMENT):

Molecule graph: w ** x -- y -- z --

Atom descriptors:

atom	type	nhs	nbrs	dots
w	carbon	3	1	0
x	carbon	2	2	0
y	nitrogen	1	2	0
z	carbon	2	2	0

RIGHT-HAND SIDE (CLEAVAGE PREDICTION):

=> Breakbond (**)

The algorithm chooses as its starting point the most general bond enviroment (x ** y) with no properties specified for either atom. During the search, this description is grown by successively specializing a property of one of the atoms in the graph or by adding a new atom to the graph. After each specialization, the new graph is checked to see if it is "better" than the parent graph from which is was derived. A daughter graph is better than its parent if it still covers at least half of the input rules (it's general enough) and still focusses on only one cleavage process (it's specific enough). The cleavage rules built by this algorithm are further improved by the program RULEMOD.

Evaluation:

i) Representational adequacy. The representation was adequate for the specific task of developing cleavage rules. It was not intended to be a general representation for objects outside of the chemical world. The descriptions can be viewed as conjunctions. Individual rules developed by the program can be considered to be linked by disjunction.

ii) Rules of generalization. The dropping condition and turning constants to variables rules are used "in reverse" during the specialization process. RULEGEN does not seem to have the ability to handle an internal disjunction but RULEMOD apparently does. For example, it can indicate that the type of atom is "anything except hydrogen". In similar work on nuclear magnetic resonance (NMR), Mitchell presents an example in which the value of nhs is listed as "greater than or equal to one" (which indicates an internal disjunction).

iii) Computational efficiency. Because this is a problem-specific algorithm, we cannot supply comparison figures here for how this algorithm would work on our test example. The current program is considered to be relatively inefficient [2].

iv) Flexibility and extensibility. Meta-DENDRAL has been extended to handle NMP spectra. The program works well in an errorful environment. It uses domain-specific knowledge extensively. However, there is no strict separation between a general-purpose induction component and a special-purpose knowledge component. It is not clear whether the methods developed for Meta-DENDRAL could be easily applied to any non-chemical domain. The program does not perform constructive induction in any general way. However, the INTSUM program does perform sophisticated transformations on the input spectra in order to develop the bond-environment descriptions.

2.3.2 Michalski and Dietterich: Program INDUCE 1.2

The algorithm described here is one of three algorithms designed by Michalski and his collaborators. The others are a data-driven method described by Stepp [21] and a mixed method described by Larson and Michalski [13,14]. The language used to describe the input events is VL_{21}, an extension to first-order predicate logic (FOPL) [17]. Each event is represented as a conjunction of selectors. A selector typically contains a function or predicate descriptor (with variables as arguments) and a list of values that the descriptor may assume. The selector [size(x1)=small, medium] asserts that the size of x1 may take the values small or medium. The events in Fig. 2 are represented as:

E1: [size(x1)=small][size(x2)=small]
[shape(x1)=circle][shape(x2)=square]
[ontop(x1,x2)]

E2: [size(x1)=small][size(x2)=large]
[size(x3)=small][shape(x1)=circle]
[shape(x2)=square][shape(x3)=circle]
[ontop(x1,x2)][inside(x3,x2)]

In this method, descriptors are divided into two classes: attribute descriptors and structure-specifying descriptors. Attribute descriptors describe attributes such as size or shape or distance which are applicable to all variables (representing, e.g., object parts). Structure-specifying descriptors include all other descriptors. They typically represent relationships among variables such as ontop or inside. Each input conjunction is broken into two conjuncts--one built of selectors containing only attribute descriptors (the attribute conjunct) and one built of selectors containing only structure-specifying descriptors (the structure conjunct).

The algorithm is based on the observation that the structure-specifying descriptors are responsible for the computational complexity of generalizing structural descriptions. If we could determine conjunctions of structure-specifying selectors which were relevant for describing a particular class of objects, then the generalization of the attribute conjuncts could be handled quickly by an appropriate covering algorithm. The algorithm seeks to determine such a set of structure conjuncts which appear likely to be part of a maximally specific conjunctive generalization of all of the in-

put events. It does this by finding conjunctions which are maximally specific generalizations of the input structure conjuncts considered alone. Such conjunctive generalizations of the structure conjuncts must be contained in some maximally specific generalizations of the entire set of input events. However, there may be maximally specific conjunctive generalizations of the input events which contain few if any structure-specifying selectors. This algorithm also finds these generalizations by considering structure conjuncts which are less than maximally specific.

The algorithm operates in two phases. The first phase is the structure-determining phase. A random sample of the input structure conjuncts is taken. This sample becomes the initial set of generalizations G_0. In each step, G_i is first pruned to a fixed size by removing unpromising generalizations. Then G_i is checked to see if any of its generalizations covers all of the structure conjuncts. If any do, they are removed from G_i and placed in the set C of candidate conjunctive generalizations. Lastly, G_i is generalized to form G_{i+1} by taking each element of G_i and generalizing it in all possible ways by dropping single selectors. When the set of candidates C reaches a prespecified size, the search stops.

The second phase is the attribute-determining phase. In this phase, the problem is converted to a multiple-valued logic covering problem using the VL_1 propositional calculus [15,16]. Each candidate cover A in C is matched against all input events and the relevant variables are identified. For each match, the appropriate attribute conjuncts are extracted and used to form a VL_1 event. For example,

if A = [ontop(p1,p2)] and
E1 = [ontop(p1,p2)][ontop(p2,p3)]
[size(p1)=1][size(p2)=3][size(p3)=5]
[color(p1)=red][color(p2)=green]
[color(p3)=blue]

then we get two VL_1 events:

V1= (1, 3, red, green) and
V2= (3, 5, green, blue).

These are vectors of attributes which correspond here to the descriptors:

(size(p1), size(p2), color(p1), color(p2))

for p1 and p2 in A.

All input events are converted into VL_1 events in this manner. In general, more than one VL_1 event is created from each input event. The set of VL_1 events can be covered using a covering algorithm. A cover could be obtained by forming the union of the values taken on by each VL_1 attribute. Such an approach usually leads to overgeneralization since only one VL_1 event derived from each input event need be covered. We use a beam-search technique to select a subset of the VL_1 events to be covered.

This two-phase algorithm provides two computational advantages. First, the time required to compare expressions in the structure-determining phase is reduced because the structure conjuncts are usually much smaller than the full input conjuncts. Second, the manipulation of VL_1 formulas is very easy since they may be represented as bit strings and manipulated using fast bit-parallel operations. The chief disadvantage of this algorithm is that it is difficult to decide when to terminate the structure-determining phase.

Evaluation:

1) Representational adequacy. The algorithm discovers, among others, the following generalizations of the events in Fig. 1:

1. [ontop(p1,p2)][size(p1)=medium]
[shape(p1)=circle,square,rectangle]
[size(p2)=large]
[shape(p2)=box,rectangle,ellipse]
[texture(p2)=clear]
There is a medium-sized circle, rectangle or square on top of a large, clear box, rectangle, or ellipse.

2. [ontop(p1,p2)][size(p1)=medium]
[shape(p1)=polygon] [texture(p1)=clear]
[size(p2)=medium,large]
[shape(p2)=rectangle,circle]
There is a clear, medium-sized polygon on top of a medium or large circle or rectangle.

3. [ontop(p1,p2)][size(p1)=medium]
[shape(p1)=polygon]
[size(p2)=medium,large]
[shape(p2)=rectangle,ellipse,circle]
There is a medium-sized polygon on top of a large or medium rectangle, ellipse or circle.

4. [size(p1)=small,medium]
[shape(p1)=circle,rectangle]
[texture(p1)=shaded]
There is a shaded object which is either medium or small in size and has a circular or rectangular shape.

This algorithm implements the conjunction, disjunction and internal disjunction operators. It provides a fairly non-uniform set of representational facilities. Descriptors, variables, and values are all distinguished. Descriptors are further analyzed into structure-specifying descriptors and attribute descriptors. The current method provides for descriptors which have unordered, linearly ordered, and tree ordered value sets. This variety of possible representations permits a better "fit" between the description language and any specific problem.

ii) Rules of generalization. The algorithm uses all rules mentioned in section I.4 and also a few constructive induction rules (see below). All constants are coded as variables. The effect of the turning-constants to variables rule is achieved as a special case of the generalization by internal disjunction rule.

iii) Computational efficiency. The algorithm requires 28 comparisons and builds 13 rules during the search to develop the descriptions listed above. Four rules are retained so this gives an efficiency ratio of 4/13 or 30%.

iv) Flexibility and extensibility. The algorithm can easily discover disjunctions by altering the termination criteria for the structure-determining phase to accept structure conjuncts which do not necessarily cover all of the input events. The same general two-phase approach can also be applied to problems of determining discriminant generalizations. Larson and Michalski have done work on determining discriminant classification rules [13,14,15].

The algorithm has good noise immunity. Noise events can be discovered because the algorithm tends to place them in separate terms of a disjunction.

Domain-specific knowledge can be incorporated into the program by defining the domains of descriptors, specifying the structures of these domains, specifying certain simple production rules, and by providing constructive induction rules. These forms of knowledge representation

are not always convenient, however. Further work should provide other facilities for knowledge representation.

A few simple constructive induction rules have been incorporated into the current implementation as a preprocessor. Other constructive induction rules can be specified by the user. Using the built-in constructive induction rules, the program produces the following conjunctive generalization of the input events in Fig. 1:

[# p's with texture clear=2][top-most(p1)]
[ontop(p1,p2)][size(p1)=medium]
[shape(p1)=polygon][texture(p1)=clear]
[size(p2)=medium,large]
[shape(p2)=circle,rectangle]

There are exactly two clear objects in each event. The top most object is a medium sized, clear polygon and it is on top of a large or medium sized circle or rectangle.

We hope to expand this constructive induction facility in the future.

2.4 Summary

The comparison of various methods is summarized in Fig. 3. The table shows the distinct advantages and disadvantages of top-down methods as opposed to bottom-up methods. Bottom-up methods tend to be faster but noise immunity and flexibility suffer as a consequence. Top-down methods have good noise immunity and are easily modified to discover disjunctive and other forms of generalization. They do tend to be computationally more expensive. By separating the structure-determining phase from the attribute-determining phase in our method, a considerable speed-up has been achieved.

3.0 CONCLUSION

One of the problems of current research on induction is that each research group is using a different formal language and terminology. This makes the exchange of information difficult. This paper was intended to help readers get a better understanding of the state of the art in this area.

Some important problems to be addressed in future research include:

i) the development of adequate formal languages and knowledge representations for hypothesis formulation and modification;

ii) extension of the scopes of operators and forms which an inductive program can efficiently use during hypothesis formulation;

iii) the development of general mechanisms of induction which can be guided by problem-specific packets of knowledge; and

iv) incorporation in the program of extensive facilities for constructive induction and multi-level schemes of description. In particular, an inductive program should be able to assign names to various subdescriptions and use these names in the formulation of hypotheses (i.e. generate hierarchical forms).

Finally, an important principle which should guide future research is what we call the principle of comprehensibility. This principle states that the descriptions which an AI program uses and the concepts which it generates should be easily comprehensible by people. In the context of work on induction, the comprehensibility principle requires that the descriptions be short and use operators which

Method: / Criterion	Hayes-Roth	Vere	Buchanan et.al.	Michalski
Intended application:	general	general	discovering mass spectrometry rules	general
Language:	Parameterized Structural Representation	Quantifier-free FOPL	Chemical model	Variable-valued logic system VL21
syntactic concepts:	case frames parameters case labels	literals constants	molecule graph attributes constants in in value sets	selectors descriptors dummy variables constants in value sets
operators:	$\wedge$	$\wedge$	$\wedge$, $\vee$, internal $\forall$	$\wedge$, $\vee$, internal $\forall$
Generalization Rules:				
dropping condition?	yes	yes	yes	yes
constants to variables?	yes	yes	yes	yes
generalizing by internal v?	no	no	yes	yes
climbing tree?	no	no	no	yes
closing intervals?	no	no	no	yes
Efficiency:				
comparisons:	22	complete algorithm not known	not applicable	28
conjunctions generated during search:	20	-----	not applicable	13
ratio output to total:	6/20=30%	-----	not applicable	4/13=30%
Extensibility:				
applications	speech analysis	none	mass spectrometry, NMR	soybean disease diagnosis
disjunctive forms?	no	yes	yes	yes
noise immunity	low	probably good	excellent	very good
domain knowledge?	no	yes	yes, built-in to program	yes
constructive induction?	no	no	no	limited facility

Figure 3.

can be easily interpreted in natural language. Furthermore, systems should be designed to provide flexible interactive facilities. This approach has been adopted in our work because we expect that the most significant applications of AI inductive programs will be as interactive tools for conceptual data analysis.

4. REFERENCES

[1] Buchanan, B. G., E. A. Feigenbaum, J. Lederberg, "A Heuristic Programming Study of Theory Formation in Science," in Proc. IJCAI-2, 1971, pp. 40-48.

[2] Buchanan, B.G., D. H. Smith, W. C. White, R. J. Gritter, E. A. Feigenbaum, J. Lederberg, C. Djerassi, J. Am. Chem. Soc. 98 (1976) p. 6168.

[3] Buchanan, B. G., E. A. Feigenbaum, "Dendral and Meta-Dendral, Their Applications Dimension," Artif. Intell. 11 (1978) pp. 5-24.

[4] Dietterich, T., "User's Guide for INDUCE1.1," internal report, Dept. of Comp. Sci., Univ. of Illnois, Urbana.

[5] Dietterich, Thomas G., "A Comparison of Methods for Characteristic Generalization," Dept. of Comp. Sci., Univ. of Ill. Rept. UIUC-DCS-78-950. Dec. 1978.

[6] Hayes-Roth, F., "Collected Papers on the Learning and Recognition of Structured Patterns", Dept. of Comp. Sci., Carnegie-Mellon Univ., Jan. 1975.

[7] Hayes-Roth, F., "Patterns of Induction and Associated Knowledge Acquisition Algorithms," Dept. of Comp. Sci., Carnegie-Mellon Univ., May 1976.

[8] Hayes-Roth, F., J. McDermott, "Knowledge Acquisition from Structure Descriptions", In Proc. IJCAI-5, 1977, pp. 356-362.

[9] Hayes-Roth, F., J. McDermott, "An Interference Matching Technique for Inducing Abstractions", CACM 21:5, 1978, pp. 401-410.

[10] Hunt, E.B., Experiments in Induction, Academic Press, 1966.

[11] Knapman, John, "A Critical Review of Winston's Learning Structural Descriptions from Examples," AISB Quarterly Issue 31, September 1978, pp. 319-320.

[12] Lenat, D., "AM: An artificial intelligence approach to discovery in mathematics as heuristic search," Comp. Sci. Dept., Rept. STAN-CS-76-570, Stanford Univ., July 1976.

[13] Larson, J., and R.S. Michalski, "Inductive Inference of VL Decision Rules," SIGART Newsletter, June 1977, pp. 38-44.

[14] Larson, J., 'Inductive Inference in the Variable Valued Predicate Logic System VL21 : Methodology and Computer Implementation', Rept. No. 869, Dept. of Comp. Sci., Univ. of Ill., Urbana, May 1977.

[15] Michalski, R. S., "Variable-valued logic and its application to pattern recognition and machine learning," In Comp. Sci. and Multiple-Valued Logic, ed. D. C. Rine, North-Holland, 1977, pp. 506-534.

[16] Michalski, R.S., "Toward Computer-aided Induction: a brief review of Currently Implemented AQVAL programs," In Proc. IJCAI-5, 1977.

[17] Michalski, R.S. "Pattern Recognition as Knowledge-Guided Induction," Rept. 927, Dept. of Comp. Sci., Univ. of Ill. Urbana, 1978.

[18] Michie, D., "New Face of AI," Experimental Programming Repts.: No. 33, MIRU, Univ. of Edinburgh, 1977.

[19] Mitchell, T. M., "Version Spaces: A Candidate Elimination Approach to Rule Learning," In Proc. IJCAI-5, MIT, 1977.

[20] Schwenzer, G. M., T. M. Mitchell, "Computer-assisted Structure Elucidation Using Automatically Acquired Carbon-13 NMR Rules," in ACS Symposium Series, No. 54, 'Computer-assisted Structure Elucidation,' D.H. Smith (ed), 1977.

[21] Stepp, R., "User's guide for UNICLASS program", internal report, Dept. of Comp. Sci., Univ. of Ill., Urbana.

[22] Vere, S.A., "Induction of Concepts in the Predicate Calculus," In Proc. IJCAI-4, 1975.

[23] Vere, S. A., "Induction of Relational Productions in the Presence of Background Information," In Proc. IJCAI-5, 1977.

[24] Vere, S. A., "Inductive Learning of Relational Productions", in Pattern-Directed Inference Systems, D.A. Waterman and F. Hayes-Roth (eds), Academic Press, 1978.

[25] Vere, S. A., "Multilevel Counterfactuals for Generalizations of Relational Concepts and Productions," Dept. of Inf. Eng'g, Univ. of Ill., Chicago Circle, 1978.